Burn

Generated by Doxygen 1.8.11

Contents

Chapter 1

burn

Daemon running on Raspberry Pi (archlinuxarm), controlling a Canberra Osprey gamma detector and a G-Star IV GPS device.

Acquisitions and measurements are merged, saved to disk and optionally transferred over a TCP connection to the controlling application, crash.

Dependencies:

- Canberra Osprey SDK V1.0.1

This software is part of a drone project at Norwegian Radiation Protection Authority (NRPA)

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 burn Namespace Reference

Namespaces

- burn
- gps_proc
- helpers
- net_proc
- proto
- spec_proc
- Utilities

6.2 burn.burn Namespace Reference

Classes

- class Burn

Variables

- filename
- level

6.2.1 Variable Documentation

6.2.1.1 burn.burn.filename

Definition at line 124 of file burn.py.

6.2.1.2 burn.burn.level

Definition at line 124 of file burn.py.

6.3 burn.gps_proc Namespace Reference

Classes

- class GpsProc

6.4 burn.helpers Namespace Reference

Functions

- def setblocking (fd, state)

6.4.1 Function Documentation

6.4.1.1 def burn.helpers.setblocking (*fd, state*)

Definition at line 22 of file helpers.py.

```
22 def setblocking(fd, state):
23     flags = fcntl.fcntl(fd, fcntl.F_GETFL)
24     if state:
25         fcntl.fcntl(fd, fcntl.F_SETFL, flags & ~os.O_NONBLOCK)
26     else:
27         fcntl.fcntl(fd, fcntl.F_SETFL, flags | os.O_NONBLOCK)
28
29
```

6.5 burn.net_proc Namespace Reference

Classes

- class NetProc

Variables

- string HOST = ''
- int PORT = 7000

6.5.1 Variable Documentation

6.5.1.1 string burn.net_proc.HOST = ''

Definition at line 25 of file net_proc.py.

6.5.1.2 int burn.net_proc.PORT = 7000

Definition at line 26 of file net_proc.py.

6.6 burn.proto Namespace Reference

Classes

- class Message

6.7 burn.spec_proc Namespace Reference

Classes

- class SpecProc

6.8 burn.Utilities Namespace Reference

Functions

- def setup ()
- def readLine (txt)
- def getStatusDescription (status)
- def getLynxAddress ()
- def getSpectralMode ()
- def getListMode ()
- def getPresetMode ()
- def getMCSPresetMode ()
- def getFloat (text, min, max)
- def getInt (text, min, max)
- def dumpException (ex)
- def reconstructAndOutputTlistData (td, timeBase, clear)
- def isLocalAddressAccessible ()
- def disableAcquisition (dtb, input)

6.8.1 Function Documentation

6.8.1.1 def burn.Utilities.disableAcquisition (*dtb, input*)

```
Description:
    This method will stop all forms of data acquisition which includes
    any of the following:
        -) SCA collection
        -) Auxiliary counter collection
        -) PHA
        -) MCS
        -) MSS
        -) DLFC
        -) List
        -) Tlist
Arguments:
    dtb (in, IDevice).  The device interface.
    input (in, int).  The input number
Return:
    none
```

Definition at line 282 of file Utilities.py.

```
282 def disableAcquisition(dtb, input):
283     """
284     Description:
285         This method will stop all forms of data acquisition which includes
286         any of the following:
287             -) SCA collection
288             -) Auxiliary counter collection
289             -) PHA
290             -) MCS
291             -) MSS
292             -) DLFC
293             -) List
294             -) Tlist
295     Arguments:
296         dtb (in, IDevice).  The device interface.
297         input (in, int).  The input number
298     Return:
299         none
300     """
301 
302     exec "from ParameterCodes import *"
303     exec "from CommandCodes import *"
304     exec "from ParameterTypes import *"
305 
306     #Make sure the input is locked before attempting any operations
307     dtb.lock("administrator", "password", input)
308 
309     #Stop acquisition
310     try:
311         dtb.control(CommandCodes.Stop, input)
312     except:
313         pass
314     #Abort acquisition (only needed for MSS or MCS collections)
315     try:
316         dtb.control(CommandCodes.Abort, input)
317     except:
318         pass
319     #Stop SCA collection
320     try:
321         dtb.setParameter(ParameterCodes.Input_SCAstatus, 0, input)
322     except:
323         pass
324     #Stop Aux counter collection
325     try:
326         dtb.setParameter(ParameterCodes.Counter_Status, 0, input)
327     except:
328         pass
329 
330 
331 
332 
```

6.8.1.2 def burn.Utilities.dumpException (*ex*)

```
Description:
    This method will print out the exception
    information
Arguments:
    ex    (in, Exception) The exception
Return:
    none
```

Definition at line 205 of file Utilities.py.

```
205 def dumpException(ex):
206     """
207     Description:
208         This method will print out the exception
209         information
210     Arguments:
211         ex    (in, Exception) The exception
212     Return:
213         none
214     """
215     print "Exception caught.  Details: %s"%str(ex)
216 
217 RolloverTime=long(0)                #This needs to be clears after a start command.
218 ROLLOVERBIT=0x00008000              #The rollover bit
219 
```

6.8.1.3 def burn.Utilities.getFloat (*text, min, max*)

```
Description:
    This method will return a floating point value
    that has been entered by the Python console
Arguments:
    text    (in, string) The text description
    min     (in, float) The min value
    max     (in, float) The max value
Return:
    (float) The value
```

Definition at line 161 of file Utilities.py.

```
161 def getFloat(text, min, max):
162     """
163     Description:
164         This method will return a floating point value
165         that has been entered by the Python console
166     Arguments:
167         text    (in, string) The text description
168         min     (in, float) The min value
169         max     (in, float) The max value
170     Return:
171         (float) The value
172     """
173     val=0.0
174     error = True
175     while error:
176         try:
177             val = readLine(text)
178             val = float(val)
179             if ((val >= min) and (val <= max)):
180                 return val
181         except:
182             pass
```

6.8.1.4 def burn.Utilities.getInt (*text, min, max*)

```
Description:
    This method will return an integer value
    that has been entered by the Python console
Arguments:
    text    (in, string) The text description
    min     (in, int) The min value
    max     (in, int) The max value
Return:
    (int) The value
```

Definition at line 183 of file Utilities.py.

```
183 def getInt(text, min, max):
184     """
185     Description:
186         This method will return an integer value
187         that has been entered by the Python console
188     Arguments:
189         text    (in, string) The text description
190         min     (in, int) The min value
191         max     (in, int) The max value
192     Return:
193         (int) The value
194     """
195     val=0
196     error = True
197     while error:
198         try:
199             val = readLine(text)
200             val = int(val)
201             if ((val >= min) and (val <= max)):
202                 return val
203         except:
204             pass
```

6.8.1.5 def burn.Utilities.getListMode ( )

```
Description:
    This method will return the list acquisition mode
    that has been entered by the Python console
Arguments:
    none
Return:
    (int) The value
```

Definition at line 96 of file Utilities.py.

```
96 def getListMode():
97     """
98     Description:
99         This method will return the list acquisition mode
100         that has been entered by the Python console
101     Arguments:
102         none
103     Return:
104         (int) The value
105     """
106     error=True
107     while error:
108         try:
109             val = readLine("Select the acquisition mode: (0=List, 1=Tlist)")
110             val = int(val)
111             if (0 == val):
112                 return 4 #List
113             elif(1 == val):
114                 return 5 #Tlist
115         except:
116             pass
```

6.8.1.6 def burn.Utilities.getLynxAddress ( )

Definition at line 72 of file Utilities.py.

```
72 def getLynxAddress():
73     return "10.0.1.4"
74
```

6.8.1.7 def burn.Utilities.getMCSPresetMode ( )

```
Description:
    This method will return the MCS preset mode
    that has been entered by the Python console
Arguments:
    none
Return:
    (int) The value
```

Definition at line 140 of file Utilities.py.

```
140 def getMCSPresetMode():
141     """
142     Description:
143         This method will return the MCS preset mode
144         that has been entered by the Python console
145     Arguments:
146         none
147     Return:
148         (int) The value
149     """
150     error=True
151     while error:
152         try:
153             val = readLine("Select the acquisition mode: (0=None, 1=Sweeps)")
154             val = int(val)
155             if (0 == val):
156                 return 0 #PresetModes.PresetNone
157             elif(1 == val):
158                 return 4 #PresetModes.PresetSweeps
159         except:
160             pass
```

6.8.1.8 def burn.Utilities.getPresetMode ( )

```
Description:
    This method will return the preset mode
    that has been entered by the Python console
Arguments:
    none
Return:
    (int) The value
```

Definition at line 117 of file Utilities.py.

```
117 def getPresetMode():
118     """
119     Description:
120         This method will return the preset mode
121         that has been entered by the Python console
122     Arguments:
123         none
124     Return:
125         (int) The value
126     """
127     error=True
128     while error:
129         try:
130             val = readLine("Select the preset mode: (0=None, 1=Real, 2=Live)")
131             val = int(val)
132             if (0 == val):
133                 return 0 #PresetModes.PresetNone
134             elif(1 == val):
135                 return 2 #PresetModes.PresetRealTime
136             elif(2 == val):
137                 return 1 #PresetModes.PresetLiveTime
138         except:
139             pass
```

6.8.1.9 def burn.Utilities.getSpectralMode ( )

```
Description:
    This method will return the spectral acquisition mode
    that has been entered by the Python console
Arguments:
    none
Return:
    (int) The value
```

Definition at line 75 of file Utilities.py.

```
75 def getSpectralMode():
76     """
77     Description:
78         This method will return the spectral acquisition mode
79         that has been entered by the Python console
80     Arguments:
81         none
82     Return:
83         (int) The value
84     """
85     error=True
86     while error:
87         try:
88             val = readLine("Select the acquisition mode: (0=Pha, 1=Dlfc)")
89             val = int(val)
90             if (0 == val):
91                 return val      #Pha
92             elif(1 == val):
93                 return 3        #Dlfc
94         except:
95             pass
```

6.8.1.10 def burn.Utilities.getStatusDescription (*status*)

```
Description:
    This method will return a string that describes the
    meaning of the various states contained in the status
    parameter.
Arguments:
    status (in, int) The status value
Return:
    (String) The description
```

Definition at line 38 of file Utilities.py.

```
38 def getStatusDescription(status):
39     """
40     Description:
41         This method will return a string that describes the
42         meaning of the various states contained in the status
43         parameter.
44     Arguments:
45         status (in, int) The status value
46     Return:
47         (String) The description
48     """
49     exec "from ParameterTypes import *"
50     statMsg="Idle "
51     if (0 != (status&StatusBits.Busy)): statMsg="Busy "
52     if (0 != (status&StatusBits.APZinprog)): statMsg+="APZ "
53     if (0 != (status&StatusBits.Diagnosing)): statMsg+="Diagnosing "
54     if (0 != (status&StatusBits.ExternalTriggerEvent)): statMsg+="Ext trig "
55     if (0 != (status&StatusBits.Fault)): statMsg+="Fault "
56     if (0 != (status&StatusBits.GroupComplete)): statMsg+="Group complete "
57     if (0 != (status&StatusBits.HVramping)): statMsg+="HVPS ramping "
58     if (0 != (status&StatusBits.Idle)): statMsg+="Idle "
59     if (0 != (status&StatusBits.PresetCompReached)): statMsg+="Comp Preset reached "
60     if (0 != (status&StatusBits.PresetTimeReached)): statMsg+="Time Preset reached "
61     if (0 != (status&StatusBits.PresetSweepsReached)): statMsg+="Sweeps Preset reached "
62     if (0 != (status&StatusBits.Rebooting)): statMsg+="Rebooting "
63     if (0 != (status&StatusBits.UpdatingImage)): statMsg+="Updating firmware "
64     if (0 != (status&StatusBits.Waiting)): statMsg+="Waiting "
65     if (0 != (status&StatusBits.AcqNotStarted)): statMsg+="Acquisition not started because preset already
       reached "
66     if (0 != (status&StatusBits.OverflowStop)): statMsg+="Acquisition stopped because channel contents
       overflowed "
67     if (0 != (status&StatusBits.ExternalStop)): statMsg+="Acquisition stopped because of external stop "
68     if (0 != (status&StatusBits.ManualStop)): statMsg+="Acquisition stopped because of manual stop "
69
70     return statMsg
71
```

6.8.1.11 def burn.Utilities.isLocalAddressAccessible ( )

```
Description:
    This method will determine whether the network address
    of the local network adapter can be obtained.
Arguments:
    none
Return:
    (bool)    True indicates that the network address can be obtained
```

Definition at line 259 of file Utilities.py.

```
259 def isLocalAddressAccessible():
260     """
261     Description:
262         This method will determine whether the network address
263         of the local network adapter can be obtained.
264     Arguments:
265         none
266     Return:
267         (bool)    True indicates that the network address can be obtained
268     """
269     try:
270         if ("Linux" == platform.system()):
271             remote = ("www.python.org", 80)
272             s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
273             s.connect( remote )
274             ip, localport = s.getsockname()
275             s.close()
276         else:
277             return True
278     except:
279         return False
280     return True
281
```

6.8.1.12 def burn.Utilities.readLine (*txt*)

```
Description:
    This method will print the text that is supplied
    to the Python console and wait for the user to
    enter a response.  The purpose is to hide the
    differences in the implementation of raw_input
    between different OS's.  Yes, there are subtle
    difference.
Arguments:
    txt  (in, string) The text to display
Return:
    (string)    The entered value
```

Definition at line 21 of file Utilities.py.

```
21 def readLine(txt):
22     """
23     Description:
24         This method will print the text that is supplied
25         to the Python console and wait for the user to
26         enter a response.  The purpose is to hide the
27         differences in the implementation of raw_input
28         between different OS's.  Yes, there are subtle
29         difference.
30     Arguments:
31         txt  (in, string) The text to display
32     Return:
33         (string)    The entered value
34     """
35     val = raw_input(txt)
36     return val.replace("\r", "")
37
```

6.8.1.13 def burn.Utilities.reconstructAndOutputTlistData (*td, timeBase, clear*)

```
Description:
    This method will reconstruct the time events for time
    stamped list mode before displaying on output
Arguments:
    td (in, TlistData).  The time stamped list data buffer.
    timeBase (in, int).  The time base (nS)
    clear (in, bool).    Resets the global time counter
Return:
    none
```

Definition at line 220 of file Utilities.py.

```
220 def reconstructAndOutputTlistData(td, timeBase, clear):
221     """
222     Description:
223         This method will reconstruct the time events for time
224         stamped list mode before displaying on output
225     Arguments:
226         td (in, TlistData).  The time stamped list data buffer.
227         timeBase (in, int).  The time base (nS)
228         clear (in, bool).    Resets the global time counter
229     Return:
230         none
231     """
232     global RolloverTime
233     if (clear): RolloverTime = long(0)
234
235     recTime=0
236     recEvent=0
237     Time=long(0)
238     conv = float(timeBase)
239     conv /= 1000 #Convert to ms
240
```

```
241     for event in td.getEvents():
242         recTime=event.getTime()
243         recEvent=event.getEvent()
244 
245         if (0 == (recTime&ROLLOVERBIT)):
246             Time = RolloverTime | (recTime & 0x7FFF)
247         else:
248             LSBofTC = int(0)
249             MSBofTC = int(0)
250             LSBofTC |= (recTime & 0x7FFF) << 15
251             MSBofTC |= recEvent << 30
252             RolloverTime = MSBofTC | LSBofTC
253 
254             #goto next event
255             continue
256         print "Event: " + str(event.getEvent()) + "; Time (uS): " + str(Time*conv)
257         Time=0
258 
```

6.8.1.14 def burn.Utilities.setup ( )

```
Description:
    This method will setup the Python package path to
    include the Lynx communications package defined
    by the \PythonExamples\DataTypes directory that came
    with the SDK CD.
Arguments:
    none
Return:
    none
```

Definition at line 6 of file Utilities.py.

```
6 def setup():
7     """
8     Description:
9         This method will setup the Python package path to
10         include the Lynx communications package defined
11         by the \PythonExamples\DataTypes directory that came
12         with the SDK CD.
13     Arguments:
14         none
15     Return:
16         none
17     """
18     toolkitPath = os.getcwd() + os.path.sep + "osprey"
19     sys.path.append(toolkitPath)
20 
```

Chapter 7

Class Documentation

7.1 burn.burn.Burn Class Reference

Public Member Functions

- def __init__ (self)
- def run (self)
- def dispatch_net_msg (self, msg)
- def dispatch_gps_msg (self, msg)
- def dispatch_spec_msg (self, msg)
- def __enter__ (self)
- def __exit__ (self, exc_type, exc_value, traceback)

Public Attributes

- running
- fdg
- fds
- fdn
- g
- s
- n

7.1.1 Detailed Description

Definition at line 31 of file burn.py.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 def burn.burn.Burn.__init__ (*self*)

Definition at line 33 of file burn.py.

```
33     def __init__(self):
34         self.running = False
35 
36         fdg_pass, self.fdg = Pipe()
37         fds_pass, self.fds = Pipe()
38         fdn_pass, self.fdn = Pipe()
39 
40         setblocking(self.fdg, 0)
41         setblocking(self.fds, 0)
42         setblocking(self.fdn, 0)
43 
44         self.g = GpsProc(fdg_pass)
45         self.s = SpecProc(fds_pass)
46         self.n = NetProc(fdn_pass)
47 
48         self.g.start()
49         self.s.start()
50         self.n.start()
51 
52         fdg_pass.close()
53         fds_pass.close()
54         fdn_pass.close()
55 
```

7.1.3 Member Function Documentation

7.1.3.1 def burn.burn.Burn.__enter__ (*self*)

Definition at line 104 of file burn.py.

```
104     def __enter__(self):
105         return self
106 
```

7.1.3.2 def burn.burn.Burn.__exit__ (*self, exc_type, exc_value, traceback*)

Definition at line 107 of file burn.py.

```
107     def __exit__(self, exc_type, exc_value, traceback):
108         self.fdg.close()
109         self.fds.close()
110         self.fdn.close()
111 
112         self.g.join()
113         self.s.join()
114         self.n.join()
115 
116         logging.info('main: terminating')
117 
```

7.1.3.3 def burn.burn.Burn.dispatch_gps_msg (*self, msg*)

Definition at line 98 of file burn.py.

```
98     def dispatch_gps_msg(self, msg):
99         self.fdn.send(msg)
100 
```

7.1.3.4 def burn.burn.Burn.dispatch_net_msg (*self, msg*)

Definition at line 75 of file burn.py.

```
75     def dispatch_net_msg(self, msg):
76         if not msg:
77             return
78         if msg.command == 'ping':
79             msg.command = 'ping_ok'
80             self.fdn.send(msg)
81         elif msg.command == 'close':
82             self.fdg.send(msg)
83             self.fds.send(msg)
84             msg.command = 'close_ok'
85             self.fdn.send(msg)
86             self.running = False
87         elif msg.command == 'new_session':
88             msg.command = 'new_session_ok'
89             msg.arguments["session_name"] = 'session1'
90             self.fdn.send(msg)
91         elif msg.command == 'get_fix':
92             self.fdg.send(msg)
93         elif msg.command == 'set_gain':
94             self.fds.send(msg)
95         elif msg.command == 'get_preview_spec':
96             self.fds.send(msg)
97
```

7.1.3.5 def burn.burn.Burn.dispatch_spec_msg (*self, msg*)

Definition at line 101 of file burn.py.

```
101     def dispatch_spec_msg(self, msg):
102         self.fdn.send(msg)
103
```

7.1.3.6 def burn.burn.Burn.run (*self*)

Definition at line 56 of file burn.py.

```
56     def run(self):
57         self.running = True
58
59         logging.info('main: warming up services')
60         time.sleep(4)
61
62         inputs = [self.fdn, self.fdg, self.fds]
63
64         while self.running:
65             readable, _, exceptional = select.select(inputs, [], inputs)
66             for s in readable:
67                 msg = s.recv()
68                 if s is self.fdn:
69                     self.dispatch_net_msg(msg)
70                 elif s is self.fdg:
71                     self.dispatch_gps_msg(msg)
72                 elif s is self.fds:
73                     self.dispatch_spec_msg(msg)
74
```

7.1.4 Member Data Documentation

7.1.4.1 burn.burn.Burn.fdg

Definition at line 36 of file burn.py.

7.1.4.2 burn.burn.Burn.fdn

Definition at line 38 of file burn.py.

7.1.4.3 burn.burn.Burn.fds

Definition at line 37 of file burn.py.

7.1.4.4 burn.burn.Burn.g

Definition at line 44 of file burn.py.

7.1.4.5 burn.burn.Burn.n

Definition at line 46 of file burn.py.

7.1.4.6 burn.burn.Burn.running

Definition at line 34 of file burn.py.

7.1.4.7 burn.burn.Burn.s

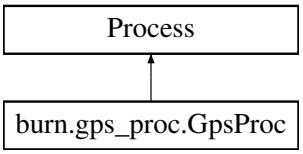
Definition at line 45 of file burn.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/burn.py

7.2 burn.gps_proc.GpsProc Class Reference

Inheritance diagram for burn.gps_proc.GpsProc:



Public Member Functions

- def __init__ (self, fd)
- def run (self)
- def dispatch (self, msg)
- def is_running (self)

Public Attributes

- fd
- gpsd
- last_lat
- last_lon
- last_alt

7.2.1 Detailed Description

Definition at line 25 of file gps_proc.py.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 def burn.gps_proc.GpsProc.__init__ (*self, fd*)

Definition at line 27 of file gps_proc.py.

```
27     def __init__(self, fd):
28         Process.__init__(self)
29         self.fd = fd
30         self.gpsd = gps(mode=WATCH_ENABLE)
31         self.last_lat = 0
32         self.last_lon = 0
33         self.last_alt = 0
34         self._running = False
35
```

7.2.3 Member Function Documentation

7.2.3.1 def burn.gps_proc.GpsProc.dispatch (*self, msg*)

Definition at line 55 of file gps_proc.py.

```
55     def dispatch(self, msg):
56         if msg.command == 'get_fix':
57             msg.command = 'get_fix_ok'
58             msg.arguments["latitude"] = self.last_lat
59             msg.arguments["longitude"] = self.last_lon
60             msg.arguments["altitude"] = self.last_alt
61             self.fd.send(msg)
62         elif msg.command == 'close':
63             self._running = False
64
```

7.2.3.2 def burn.gps_proc.GpsProc.is_running (*self*)

Definition at line 65 of file gps_proc.py.

```
65     def is_running(self):
66         return self._running
67
```

7.2.3.3 def burn.gps_proc.GpsProc.run (*self*)

Definition at line 36 of file gps_proc.py.

```
36     def run(self):
37         self._running = True
38         logging.info('gpsd: starting service')
39 
40         while self._running:
41             while self.gpsd.waiting():
42                 self.gpsd.next()
43                 if not math.isnan(self.gpsd.fix.latitude):
44                     self.last_lat = self.gpsd.fix.latitude
45                 if not math.isnan(self.gpsd.fix.longitude):
46                     self.last_lon = self.gpsd.fix.longitude
47                 if not math.isnan(self.gpsd.fix.altitude):
48                     self.last_alt = self.gpsd.fix.altitude
49 
50             while self.fd.poll():
51                 self.dispatch(self.fd.recv())
52 
53         logging.info('gpsd: terminating')
54 
```

7.2.4 Member Data Documentation

7.2.4.1 burn.gps_proc.GpsProc.fd

Definition at line 29 of file gps_proc.py.

7.2.4.2 burn.gps_proc.GpsProc.gpsd

Definition at line 30 of file gps_proc.py.

7.2.4.3 burn.gps_proc.GpsProc.last_alt

Definition at line 33 of file gps_proc.py.

7.2.4.4 burn.gps_proc.GpsProc.last_lat

Definition at line 31 of file gps_proc.py.

7.2.4.5 burn.gps_proc.GpsProc.last_lon

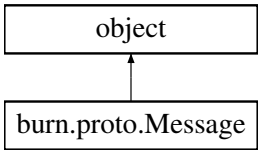
Definition at line 32 of file gps_proc.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/gps_proc.py

7.3 burn.proto.Message Class Reference

Inheritance diagram for burn.proto.Message:



Public Member Functions

- def __init__ (self, command='', arguments={})

Public Attributes

- command
- arguments

7.3.1 Detailed Description

Definition at line 20 of file proto.py.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 def burn.proto.Message.__init__ (*self, command = '', arguments = {}*)

Definition at line 21 of file proto.py.

```
21     def __init__(self, command='', arguments={}):
22         self.command = command
23         self.arguments = arguments
24
```

7.3.3 Member Data Documentation

7.3.3.1 burn.proto.Message.arguments

Definition at line 23 of file proto.py.

7.3.3.2 burn.proto.Message.command

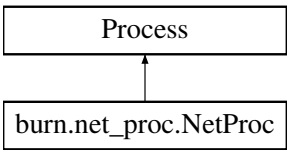
Definition at line 22 of file proto.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/proto.py

7.4 burn.net_proc.NetProc Class Reference

Inheritance diagram for burn.net_proc.NetProc:



Public Member Functions

- def __init__ (self, fd)
- def run (self)
- def dispatch_msg (self)
- def is_running (self)

Public Attributes

- fd
- addr
- sock
- buffer

7.4.1 Detailed Description

Definition at line 28 of file net_proc.py.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 def burn.net_proc.NetProc.__init__ (*self, fd*)

```
Initialization of the net process
```

Definition at line 30 of file net_proc.py.

```
30     def __init__(self, fd):
31         """
32         Initialization of the net process
33         """
34         Process.__init__(self)
35         self.fd = fd
36         setblocking(self.fd, 0)
37         self._running = False
38         self.conn, self.addr = None, None
39         self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
40         self.sock.setblocking(0)
41         self.buffer = ''
42         try:
43             self.sock.bind((HOST, PORT))
44         except socket.error as e:
45             logging.error('network: bind failed: ' + os.strerror(e.errno))
46 
47         self.sock.listen(5)
48         logging.info('network: service listening')
49 
```

7.4.3 Member Function Documentation

7.4.3.1 def burn.net_proc.NetProc.dispatch_msg (*self*)

```
Convert received data to messages and pass them on to main controller
```

Definition at line 120 of file net_proc.py.

```
120     def dispatch_msg(self):
121         """
122         Convert received data to messages and pass them on to main controller
123         """
124         while True:
125             if len(self.buffer) < 4:
126                 return
127             # Extract message length
128             msglen = struct.unpack("!I", self.buffer[0:4])[0]
129             if len(self.buffer) < msglen+4:
130                 logging.info('network: buffer not ready')
131                 return
132             # Extract rest of message and convert to object
133             jmsg = json.loads(self.buffer[4:4+msglen])
134             msg = Message(**jmsg)
135             # Pass message to main controller
136             self.fd.send(msg)
137             # Update buffer
138             self.buffer = self.buffer[4+msglen:]
139
```

7.4.3.2 def burn.net_proc.NetProc.is_running (*self*)

```
Return wether the net process is still running
```

Definition at line 140 of file net_proc.py.

```
140     def is_running(self):
141         """
142         Return wether the net process is still running
143         """
144         return self._running
145
```

7.4.3.3 def burn.net_proc.NetProc.run (*self*)

```
Entry point for the net process
```

Definition at line 50 of file net_proc.py.

```
50     def run(self):
51         """
52         Entry point for the net process
53         """
54         logging.info('network: starting service')
55         self._running = True
56         # Prepare sockets and file descriptors
57         inputs = [self.fd, self.sock]
58
59         # Start select event loop
60         while(self._running):
61
62             readable, _, _ = select.select(inputs, [], [])
63
```

```
64              for s in readable: # Handle reads
65
66                  if s is self.sock: # Incoming connection on listening socket
67                      # We only allow one connection at a time (TODO)
68                      self.conn, self.addr = s.accept()
69                      self.conn.setblocking(0)
70                      inputs.append(self.conn)
71                      self.buffer = ''
72                      logging.info('network: connection received from ' + self.
    addr[0])
73
74                  elif s is self.fd: # Incoming message from main controller
75                      msg = s.recv()
76                      data = json.dumps(msg.__dict__) # Convert object to json
77                      netstring = struct.pack("!I", len(data)) + data # Serialize json
78                      totlen, currlen = len(netstring), 0
79                      while True:
80                          # Send complete packet
81                          l = self.conn.send(netstring[currlen:])
82                          if l == 0:
83                              inputs.remove(self.conn)
84                              self.conn.close()
85                              logging.info('network: connection broken from ' + self.
    addr[0])
86                              break
87                          currlen += l
88                          if currlen >= totlen:
89                              break
90                      if msg.command == 'close_ok': # main controller is closing
91                          self._running = False
92
93                  else: # Incoming data from existing connection
94                      try:
95                          data = s.recv(1024)
96                      except socket.error as e:
97                          if e.errno == errno.ECONNRESET:
98                              inputs.remove(s)
99                              s.close()
100                         logging.error('network: ' + self.addr[0] + ': ' + os.strerror(e.errno))
101                         continue
102                     if not data or data == '':
103                         inputs.remove(s)
104                         s.close()
105                         logging.error('network: connection lost')
106                         continue
107                     else:
108                         # Data successfully received, store in buffer
109                         self.buffer += data
110                         self.dispatch_msg()
111
112         # Close active connections
113         if self.conn is not None:
114             self.conn.close()
115         if self.sock is not None:
116             self.sock.close()
117
118         logging.info('network: terminating')
119
```

7.4.4 Member Data Documentation

7.4.4.1 burn.net_proc.NetProc.addr

Definition at line 38 of file net_proc.py.

7.4.4.2 burn.net_proc.NetProc.buffer

Definition at line 41 of file net_proc.py.

7.4.4.3 burn.net_proc.NetProc.fd

Definition at line 35 of file net_proc.py.

7.4.4.4 burn.net_proc.NetProc.sock

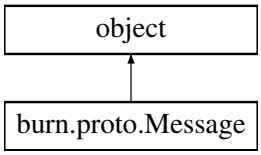
Definition at line 39 of file net_proc.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/net_proc.py

7.5 object Class Reference

Inheritance diagram for object:

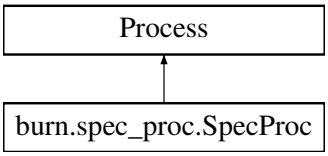


The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/proto.py

7.6 burn.spec_proc.SpecProc Class Reference

Inheritance diagram for burn.spec_proc.SpecProc:



Public Member Functions

- def __init__ (self, fd)
- def run (self)
- def dispatch (self, msg)
- def reset_acquisition (self)
- def stabilize_probe (self, voltage, coarse_gain, fine_gain)
- def run_preview (self, msg)

Public Attributes

- fd
- running
- source_dir
- group
- input
- dtb

7.6.1 Detailed Description

Definition at line 38 of file spec_proc.py.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 def burn.spec_proc.SpecProc.__init__ (*self, fd*)

Definition at line 39 of file spec_proc.py.

```
39     def __init__(self, fd):
40         Process.__init__(self)
41         self.fd = fd
42         self.running = False
43         self.source_dir = ""
44         self.group = 1
45         self.input = 1
46         self.dtb = DeviceFactory.createInstance(DeviceFactory.DeviceInterface.IDevice)
47         self.dtb.open("", Utilities.getLynxAddress())
48         logging.info('spec: using device ' + self.dtb.getParameter(ParameterCodes.Network_MachineName, 0))
49         self.dtb.lock("administrator", "password", self.input)
50
```

7.6.3 Member Function Documentation

7.6.3.1 def burn.spec_proc.SpecProc.dispatch (*self, msg*)

Definition at line 59 of file spec_proc.py.

```
59     def dispatch(self, msg):
60         if msg.command == 'set_gain':
61             voltage = msg.arguments["voltage"]
62             coarse = msg.arguments["coarse_gain"]
63             fine = msg.arguments["fine_gain"]
64             self.stabilize_probe(voltage, coarse, fine)
65             logging.info('spec: gain has been set')
66             msg.command = 'set_gain_ok'
67             self.fd.send(msg)
68         elif msg.command == 'close':
69             self.running = False
70         elif msg.command == 'get_preview_spec':
71             self.reset_acquisition()
72             self.run_preview(msg)
73             msg.command = 'get_preview_spec_ok'
74             self.fd.send(msg)
75         else:
76             logging.warning('spec: unknown command ' + cmd.command)
77
```

7.6.3.2 def burn.spec_proc.SpecProc.reset_acquisition (*self*)

Definition at line 78 of file spec_proc.py.

```
78     def reset_acquisition(self):
79         #Disable all acquisition
80         Utilities.disableAcquisition(self.dtb, self.input)
81         #Set the acquisition mode. The Only Available Spectral in Osprey is Pha = 0
82         self.dtb.setParameter(ParameterCodes.Input_Mode, 0, self.input)
83         #Setup presets
84         self.dtb.setParameter(ParameterCodes.Preset_Options, 1, self.input)
85         #Clear data and time
86         self.dtb.control(CommandCodes.Clear, self.input)
87         #Set the current memory group
88         self.dtb.setParameter(ParameterCodes.Input_CurrentGroup, self.group, self.
       input)
89
```

7.6.3.3 def burn.spec_proc.SpecProc.run (*self*)

Definition at line 51 of file spec_proc.py.

```
51     def run(self):
52         logging.info('spec: staring service')
53         self.running = True
54         while(self.running):
55             if self.fd.poll():
56                 self.dispatch(self.fd.recv())
57         logging.info('spec: terminating')
58
```

7.6.3.4 def burn.spec_proc.SpecProc.run_preview (*self, msg*)

Definition at line 107 of file spec_proc.py.

```
107     def run_preview(self, msg):
108         livetime = msg.arguments["livetime"]
109         # Setup presets
110         self.dtb.setParameter(ParameterCodes.Preset_Live, float(livetime), self.
    input)
111         # Clear data and time
112         self.dtb.control(CommandCodes.Clear, self.input)
113         # Start the acquisition
114         self.dtb.control(CommandCodes.Start, self.input)
115         while True:
116             sd = self.dtb.getSpectralData(self.input, self.group)
117             if ((0 == (StatusBits.Busy & sd.getStatus())) and (0 == (StatusBits.Waiting & sd.getStatus()))
    :
118                 break
119             time.sleep(.1)
120
121         chans = sd.getSpectrum().getCounts()
122         total_count = 0
123         channel_string = ''
124         for ch in chans:
125             total_count += ch
126             channel_string += str(ch) + ' '
127
128         msg.arguments["channels"] = channel_string.strip()
129         msg.arguments["channel_count"] = len(chans)
130         msg.arguments["uncorrected_total_count"] = total_count
131         msg.arguments["livetime"] = sd.getLiveTime()
132         msg.arguments["realtime"] = sd.getRealTime()
133         msg.arguments["computational_limit"] = sd.getComputationalValue()
134         msg.arguments["status"] = Utilities.getStatusDescription(sd.getStatus())
135         """print "Input: %d; Group: %d"%(sd.getInput(), sd.getGroup())"""
136         #self.save(sd, i)
137
138 #    def create_session(self, acquisition_interval, acquisition_spacing, acquisition_count):
139 #        self.acquisition_interval = acquisition_interval
140 #        self.acquisition_spacing = acquisition_spacing
141 #        self.acquisition_count = acquisition_count
142 #
143 #        now = datetime.now()
144 #        self.source_dir = os.path.expanduser("/tmp/ashes/") + now.strftime("%Y%m%d_%H%M%S")
145 #        os.makedirs(self.source_dir, 0777)
146 #        print self.source_dir
147 #        #Set the current memory group
148 #        self.dtb.setParameter(ParameterCodes.Input_CurrentGroup, self.group, self.input)
149
150 #    def save(self, sd, idx):
151 #        chans = sd.getSpectrum().getCounts()
152 #        mca, sec, rt, lt, dat, tim, off, nc = 1, 0, sd.getRealTime(), sd.getLiveTime(), "07DEC151",
    "0707", 0, len(chans) # FIXME
153 #        hdr = pack("hhhhii8s4shh", -1, mca, 1, sec, rt, lt, dat, tim, off, nc)
154 #        with open(self.source_dir + os.path.sep + str(idx) + ".chn", "w+b") as f:
155 #            f.write(hdr)
156 #            int_array = array('L', chans)
157 #            int_array.tofile(f)
158 #            f.close()
159
```

7.6.3.5 def burn.spec_proc.SpecProc.stabilize_probe (*self, voltage, coarse_gain, fine_gain*)

Definition at line 90 of file spec_proc.py.

```
90     def stabilize_probe(self, voltage, coarse_gain, fine_gain):
91         # Turn on HV
92         Stabilized_Probe_Bussy = 0x00080000
93         Stabilized_Probe_OK = 0x00100000
94         dtb_probe_type = self.dtb.getParameter(ParameterCodes.Input_Status, self.
       input)
95         if((dtb_probe_type & Stabilized_Probe_OK) != Stabilized_Probe_OK):
96             #HV_Value = Utilities.readLine("Enter HV Value: ")
97             self.dtb.setParameter(ParameterCodes.Input_Voltage, int(voltage), self.
       input)
98             self.dtb.setParameter(ParameterCodes.Input_VoltageStatus, True, self.
       input)
99             #Wait till ramping is complete
100             logging.info('spec: ramping HVPS...')
101             while(self.dtb.getParameter(ParameterCodes.Input_VoltageRamping, self.
       input) is True):
102                 time.sleep(.4)
103         # Set gain
104         self.dtb.setParameter(ParameterCodes.Input_CoarseGain, float(coarse_gain), self.
       input) # [1.0, 2.0, 4.0, 8.0]
105         self.dtb.setParameter(ParameterCodes.Input_FineGain, float(fine_gain), self.
       input) # [1.0, 5.0]
106
```

7.6.4 Member Data Documentation

7.6.4.1 burn.spec_proc.SpecProc.dtb

Definition at line 46 of file spec_proc.py.

7.6.4.2 burn.spec_proc.SpecProc.fd

Definition at line 41 of file spec_proc.py.

7.6.4.3 burn.spec_proc.SpecProc.group

Definition at line 44 of file spec_proc.py.

7.6.4.4 burn.spec_proc.SpecProc.input

Definition at line 45 of file spec_proc.py.

7.6.4.5 burn.spec_proc.SpecProc.running

Definition at line 42 of file spec_proc.py.

7.6.4.6 burn.spec_proc.SpecProc.source_dir

Definition at line 43 of file spec_proc.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/spec_proc.py

Chapter 8

File Documentation

8.1 /home/drb/dev/py/burn/__init__.py File Reference

Namespaces

- burn

8.2 /home/drb/dev/py/burn/burn.py File Reference

Classes

- class burn.burn.Burn

Namespaces

- burn.burn

Variables

- burn.burn.filename
- burn.burn.level

8.3 /home/drb/dev/py/burn/gps_proc.py File Reference

Classes

- class burn.gps_proc.GpsProc

Namespaces

- burn.gps_proc

8.4 /home/drb/dev/py/burn/helpers.py File Reference

Namespaces

- burn.helpers

Functions

- def burn.helpers.setblocking (fd, state)

8.5 /home/drb/dev/py/burn/net_proc.py File Reference

Classes

- class burn.net_proc.NetProc

Namespaces

- burn.net_proc

Variables

- string burn.net_proc.HOST = ''
- int burn.net_proc.PORT = 7000

8.6 /home/drb/dev/py/burn/proto.py File Reference

Classes

- class burn.proto.Message

Namespaces

- burn.proto

8.7 /home/drb/dev/py/burn/README.md File Reference

8.8 /home/drb/dev/py/burn/spec_proc.py File Reference

Classes

- class burn.spec_proc.SpecProc

Namespaces

- burn.spec_proc

8.9 /home/drb/dev/py/burn/Utilities.py File Reference

Namespaces

- burn.Utilities

Functions

- def burn.Utilities.setup ()
- def burn.Utilities.readLine (txt)
- def burn.Utilities.getStatusDescription (status)
- def burn.Utilities.getLynxAddress ()
- def burn.Utilities.getSpectralMode ()
- def burn.Utilities.getListMode ()
- def burn.Utilities.getPresetMode ()
- def burn.Utilities.getMCSPresetMode ()
- def burn.Utilities.getFloat (text, min, max)
- def burn.Utilities.getInt (text, min, max)
- def burn.Utilities.dumpException (ex)
- def burn.Utilities.reconstructAndOutputTlistData (td, timeBase, clear)
- def burn.Utilities.isLocalAddressAccessible ()
- def burn.Utilities.disableAcquisition (dtb, input)

Index